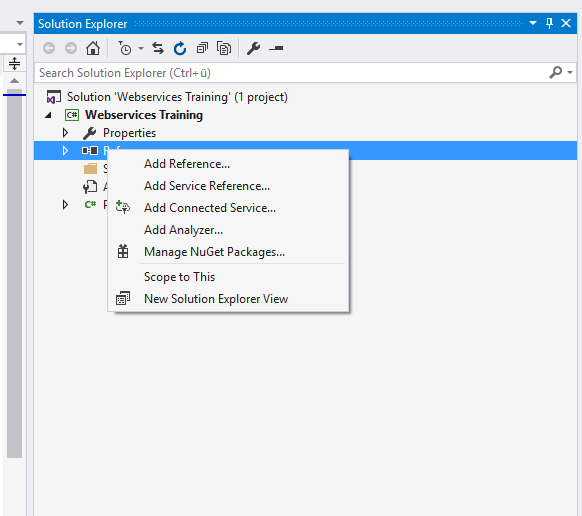
# Webservices

## Voraussetzungen

* Das HTTPS-Protokoll sollte aktiviert sein.

## Projekt Erstellen

* Öffnen Sie Microsoft Visual Studio und erstellen Sie ein neues “Console Application“-Projekt. Das Projekt sollte im Ordner “Übung“ gespeichert werden. Benennen Sie das Projekt “Webservice Training“.
* Machen Sie Rechtklick auf die “References“ im “Solution Explorer“ und selektieren Sie die Option “Add Service Reference“.



* Verwenden Sie die folgende Addresse für die Webservices:

<https://demo.neplan.ch/NEPLAN360_Demo/Services/External/NeplanService.svc>

* Verwenden Sie als “Namespace“ die Bezeichnung “NeplanService“ und drücken Sie “OK“.

## Webservice-Klasse

* Definieren Sie folgende Import-Bibliotheken:

using Webservices\_Training.NeplanService;

using System.Security.Cryptography;

using System.Xml;

using System.Globalization;

* Erstellen Sie eine neue Klasse mit dem Namen “Webservice“ wie in Kapitel 2.1 dargestellt.
* Der Variable „project“ sollte der Name “Neplan Beispiel“ zugewiesen werden. Das Projekt sollte in der Datenbank vorhanden sein (es sollte vorher importiert worden sein).
* Füllen Sie “username“ und “password“ aus, welche Sie von der Lehrperson erhalten.
* Ergänzen Sie den folgende Code in “Main“:

Webservice web = new Webservice();

Console.WriteLine("Press any key to stop...");

Console.ReadKey();

* Starten Sie die Applikation

## Klasse „Program“

* Ersetzen Sie die Klasse „Program“ mit dem Code in 2.2.

## Lastfluss-Berechnung

* Ersetzen Sie die Funktion “RunLoadFlow“ mit dem in Kapitel 2.3 angegebenen Code. Diese Funktion ruft die LastFluss-Berechnung auf. Nach der Berechnung werden die Resultate in der Datenbank gespeichert.

## Lastfluss Ergebnisse

* Ersetzen Sie die Funktion “GetResultsLoadFlow“ mit dem Code in 2.4 und starten Sie die Applikation. Nach der Berechnung werden sowohl die Verluste als auch die Wirkleistung der Netzeinspeisung geholt und dargestellt.

## Schalterzustand Änderung

* Verwenden Sie den Code in 2.5 um die eine Last des Beispiels abzuschalten.

## Attributeänderung

* Zunächst ändern Sie die Produktion der PV-Installation in dem Sie den Code in 2.6 benützen. Die Produktion wird auf 40 kW verändert.
* Die Code in Kapitel 2.6 enthält eine Methode um die Lastflussberechnung zu wiederholen und die Resultate nochmals aufzurufen.

## Default-Werte

* Die Attribute und der Schalterzustand werden direkt in der Datenbank geändert. Aus diesem Grund sollten in diesem Beispiel, nach der Berechnung wieder die Default-Werte gesetzt werden.
* Kopieren Sie den Code in Kapitel 2.7 in das Programm, um das Beispiel in den ursprünglichen Zustand zu versetzten.
* Starten Sie die Applikation und überprüfen Sie die Richtung der Wirkleistung nach den in den Kapiteln 1.8 und 1.9 vorgenommenen Veränderungen.

## Trennstellenoptimierung

* Eine Trennstellenoptimierung kann ähnlich wie eine Lastfluss-Berechnung ausgeführt werden. Den entsprechenden Code können Sie in Kapitel 2.8 finden.

## Trennstellenoptimierung: Ergebnisse

* Der Code in Kapitel 2.9 definiert, wie Sie die Elementresultate nach der Trennstelloptimierung erhalten können.
* Kopieren Sie den Code in den Kapiteln 2.8 und 2.9 in ihr Programm und starten Sie die Applikation neu.

# ANHANG

#### Code der Webservice-Klasse

public class Webservice

{

public NeplanServiceClient nepService;

public ExternalProject ext;

private string username = "";

private string password = "";

public string project = "Neplan Beispiel";

public Webservice()

{

nepService = new NeplanServiceClient(); //instantiates the neplaservice

nepService.ClientCredentials.UserName.UserName = username; //give the username

nepService.ClientCredentials.UserName.Password = getMd5Hash(password); //give the password

try

{

nepService.Open(); //open the service

Console.WriteLine("Opened service");

ext = nepService.GetProject(project, null, null, null); //get the project

if (ext != null)

Console.WriteLine("Got project");

else

Console.WriteLine("Cannot get project");

}

catch

{

Console.WriteLine("Cannot open service");

}

}

public void CloseWebservice()

{

try

{

nepService.Close(); //close the service

Console.WriteLine("Closed service");

}

catch

{

Console.WriteLine("Cannot close service");

}

}

private static string getMd5Hash(string input)

{

#region Not important for the training

// Create a new instance of the MD5CryptoServiceProvider object.

MD5 md5Hasher = MD5.Create();

// Convert the input string to a byte array and compute the hash.

byte[] data = md5Hasher.ComputeHash(Encoding.Default.GetBytes(input));

// Create a new Stringbuilder to collect the bytes

// and create a string.

StringBuilder sBuilder = new StringBuilder();

// Loop through each byte of the hashed data

// and format each one as a hexadecimal string.

for (int i = 0; i < data.Length; i++)

sBuilder.Append(data[i].ToString("x2"));

// Return the hexadecimal string.

return sBuilder.ToString();

#endregion

}

}

#### Code der Klasse „Program“

class Program

{

bool CalcOk = true;

static void Main(string[] args)

{

Webservice webservice = new Webservice();

Program example = new Program();

if (webservice.nepService != null && webservice.ext != null) //Checks if the interface was created

{

example.RunLoadFlow(webservice); //runs a load flow

example.GetResultsLoadFlow(webservice); //gets results

example.OpenSwitch(webservice); //open a switch

example.ChangePsetting(webservice); //sets P of a 1-port element

example.RepeatLoadFLowandResults(webservice); //repeats the process

example.RestoreOriginalValues(webservice);

example.RunSwitchingOptimisation(webservice);

example.GetResultsSwitchingOptimization(webservice);

}

webservice.CloseWebservice();

Console.WriteLine("Press any key to stop...");

Console.ReadKey();

}

internal void RunLoadFlow(Webservice webservice)

{

}

internal void GetResultsLoadFlow(Webservice webservice)

{

}

internal void OpenSwitch(Webservice webservice)

{

}

internal void ChangePsetting(Webservice webservice)

{

}

internal void RepeatLoadFLowandResults(Webservice webservice)

{

}

internal void RestoreOriginalValues(Webservice webservice)

{

}

internal void RunSwitchingOptimisation(Webservice webservice)

{

}

internal void GetResultsSwitchingOptimization(Webservice webservice)

{

}

/// <summary>

/// gets the value of an xml element

/// </summary>

/// <param name="result"></param>

/// <param name="xmlelement"></param>

/// <returns></returns>

private string GetXMLAttribute(string result, string xmlelement)

{

if (string.IsNullOrEmpty(result))

return null;

XmlDocument xmldoc = new XmlDocument();

xmldoc.LoadXml(result);

XmlNodeList nodeList = xmldoc.GetElementsByTagName(xmlelement);

if (nodeList == null || nodeList.Count != 1)

{

CalcOk = false;

return null;

}

else

return nodeList[0].InnerText;

}

}

#### Code für die Lastflussberechnung

internal void RunLoadFlow(Webservice webservice)

{

if (!CalcOk)

return;

//run load flow. It is possible to define the operational state

AnalysisReturnInfo analysis = webservice.nepService.AnalyseVariant(webservice.ext, Guid.NewGuid().ToString(), "LoadFlow", "Default", string.Empty, string.Empty, string.Empty);

if (analysis.ReturnInfo == 1 && analysis.HasConverged) //returninfo should be 1 if the calculation was done successfully. HasConverged is a special flag only to be used for Load Flow

Console.WriteLine("Load Flow run successfully!");

else

{

Console.WriteLine("Could not run Load Flow!");

CalcOk = false;

}

}

#### Code für das Holen der Ergebnisse des Lastflusses

internal void GetResultsLoadFlow(Webservice webservice)

{

if (!CalcOk)

return;

int networkTypeGroup = 0; //identifier for network results (other indexes refer to area, zone, feeder etc.)

string[] networkresults = webservice.nepService.GetListResultSummary(webservice.ext, "LoadFlow", new DateTime(), networkTypeGroup, null);

if (networkresults == null || networkresults.Count() != 1)

{

CalcOk = false;

Console.WriteLine("Could not get the network results!");

return;

}

else

{

string plosses = GetXMLAttribute(networkresults[0], "PLosses");

string qlosses = GetXMLAttribute(networkresults[0], "QLosses");

Console.WriteLine(string.Format("Network Active Losses: {0:0.000}kW", Convert.ToDouble(plosses, CultureInfo.InvariantCulture) \* 1000));

Console.WriteLine(string.Format("Network Reactive Losses: {0:0.000}kVar", Convert.ToDouble(qlosses, CultureInfo.InvariantCulture) \* 1000));

}

int portNumber = 0; //0 is the first port connection of an element in Neplan

string elementname = "Netz";

string elementtype = "ExternalGrid";

//gets the element results for the external grid

string result = webservice.nepService.GetResultElementByName(webservice.ext, elementname, elementtype, portNumber, "LoadFlow", new DateTime());

if (result == null)

{

CalcOk = false;

Console.WriteLine("Could not get the results of ");

return;

}

else

{

string power = GetXMLAttribute(result, "P"); //gets the power import for the network

Console.WriteLine(string.Format("The network is supplied with: {0:0.00} kW", -Convert.ToDouble(power, CultureInfo.InvariantCulture) \* 1000));

}

}

#### Code für die Änderung des Schalterzustandes

internal void OpenSwitch(Webservice webservice)

{

if (!CalcOk)

return;

string elementname = "Verbraucher A";

string elementtype = "Load";

short portNumber = 0;

webservice.nepService.SwitchElementAtPort(webservice.ext, elementname, elementtype, portNumber, false); //switch out one of the loads

Console.WriteLine(string.Format("{0} diconnected", elementname));

}

#### Code Änderung der Attribute

internal void ChangePsetting(Webservice webservice)

{

if (!CalcOk)

return;

string elementname = "PV Installation";

string elementtype = "ACDisperseGenerator";

string attributename = "Pset";

string power = "40";

if (webservice.nepService.SetElementAttribute(webservice.ext, elementname, elementtype, attributename, power)) //set the PV Installation production to 40kW

Console.WriteLine(string.Format("Output of {0} set to {1}kW", elementname, power));

else

{

CalcOk = false;

Console.WriteLine("Could not change output of " + elementname);

}

}

internal void RepeatLoadFLowandResults(Webservice webservice)

{

if (!CalcOk)

return;

RunLoadFlow(webservice);

GetResultsLoadFlow(webservice);

}

#### Definition der Default-Werte

internal void RestoreOriginalValues(Webservice webservice)

{

if (!CalcOk)

return;

string elementname = "PV Installation";

string elementtype = "ACDisperseGenerator";

string attributename = "Pset";

string power = "4";

webservice.nepService.SetElementAttribute(webservice.ext, elementname, elementtype, attributename, power);

elementname = "Verbraucher A";

elementtype = "Load";

short portNumber = 0;

webservice.nepService.SwitchElementAtPort(webservice.ext, elementname, elementtype, portNumber, true);

Console.WriteLine("Restored original values of the example");

}

#### Code Trennstellenoptimierung

internal void RunSwitchingOptimisation(Webservice webservice)

{

if (!CalcOk)

return;

webservice.ext = webservice.nepService.GetProject("SwitchingOpimization", null, null, null); //get the project

if (webservice.ext == null)

Console.WriteLine("Cannot get SwitchingOptimization project");

//run switching optimisation.

AnalysisReturnInfo analysis = webservice.nepService.AnalyseVariant(webservice.ext, Guid.NewGuid().ToString(), "SwitchingOptimization", "Default", string.Empty, string.Empty, string.Empty);

if (analysis.ReturnInfo == 1)

Console.WriteLine("Switching Optimisation run successfully!");

else

{

Console.WriteLine("Could not run Switching Optimisation!");

CalcOk = false;

}

}

#### Code für das Holen der Ergebnisse der Trennstellenoptimierung

internal void GetResultsSwitchingOptimization(Webservice webservice)

{

if (!CalcOk)

return;

var results = webservice.nepService.GetAllElementResults(webservice.ext, "SwitchingOptimization"); //get the results of switching optimisation

if (results != null)

foreach (var res in results)

Console.WriteLine(string.Format("Element {0} was switched {1} on side {2}", res.Name, GetXMLAttribute(res.XMLdata, "SwitchFinal"), res.portNr));

}

#### Komplettes Beispiel (Program.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Webservices\_Training.NeplanService;

using System.Security.Cryptography;

using System.Xml;

using System.Globalization;

namespace Webservices\_Training

{

class Program

{

bool CalcOk = true;

static void Main(string[] args)

{

Webservice webservice = new Webservice();

Program example = new Program();

if (webservice.nepService != null && webservice.ext != null) //Checks if the interface was created

{

example.RunLoadFlow(webservice); //runs a load flow

example.GetResultsLoadFlow(webservice); //gets results

example.OpenSwitch(webservice); //open a switch

example.ChangePsetting(webservice); //sets P of a 1-port element

example.RepeatLoadFLowandResults(webservice); //repeats the process

example.RestoreOriginalValues(webservice);

example.RunSwitchingOptimisation(webservice);

example.GetResultsSwitchingOptimization(webservice);

}

webservice.CloseWebservice();

Console.WriteLine("Press any key to stop...");

Console.ReadKey();

}

internal void RunLoadFlow(Webservice webservice)

{

if (!CalcOk)

return;

//run load flow. It is possible to define the operational state

AnalysisReturnInfo analysis = webservice.nepService.AnalyseVariant(webservice.ext, Guid.NewGuid().ToString(), "LoadFlow", "Default", string.Empty, string.Empty, string.Empty);

if (analysis.ReturnInfo == 1 && analysis.HasConverged) //returninfo should be 1 if the calculation was done successfully. HasConverged is a special flag only to be used for Load Flow

Console.WriteLine("Load Flow run successfully!");

else

{

Console.WriteLine("Could not run Load Flow!");

CalcOk = false;

}

}

internal void GetResultsLoadFlow(Webservice webservice)

{

if (!CalcOk)

return;

int networkTypeGroup = 0; //identifier for network results (other indexes refer to area, zone, feeder etc.)

string[] networkresults = webservice.nepService.GetListResultSummary(webservice.ext, "LoadFlow", new DateTime(), networkTypeGroup, null);

if (networkresults == null || networkresults.Count() != 1)

{

CalcOk = false;

Console.WriteLine("Could not get the network results!");

return;

}

else

{

string plosses = GetXMLAttribute(networkresults[0], "PLosses");

string qlosses = GetXMLAttribute(networkresults[0], "QLosses");

Console.WriteLine(string.Format("Network Active Losses: {0:0.000}kW", Convert.ToDouble(plosses, CultureInfo.InvariantCulture) \* 1000));

Console.WriteLine(string.Format("Network Reactive Losses: {0:0.000}kVar", Convert.ToDouble(qlosses, CultureInfo.InvariantCulture) \* 1000)); }

int portNumber = 0; //0 is the first port connection of an element in Neplan

string elementname = "Netz";

string elementtype = "ExternalGrid";

//gets the element results for the external grid

string result = webservice.nepService.GetResultElementByName(webservice.ext, elementname, elementtype, portNumber, "LoadFlow", new DateTime());

if (result == null)

{

CalcOk = false;

Console.WriteLine("Could not get the results of ");

return;

}

else

{

string power = GetXMLAttribute(result, "P"); //gets the power import for the network

Console.WriteLine(string.Format("The network is supplied with: {0:0.00} kW", -Convert.ToDouble(power, CultureInfo.InvariantCulture) \* 1000));

}

}

internal void OpenSwitch(Webservice webservice)

{

if (!CalcOk)

return;

string elementname = "Verbraucher A";

string elementtype = "Load";

short portNumber = 0;

webservice.nepService.SwitchElementAtPort(webservice.ext, elementname, elementtype, portNumber, false); //switch out one of the loads

Console.WriteLine(string.Format("{0} diconnected", elementname));

}

internal void ChangePsetting(Webservice webservice)

{

if (!CalcOk)

return;

string elementname = "PV Installation";

string elementtype = "ACDisperseGenerator";

string attributename = "Pset";

string power = "40";

if (webservice.nepService.SetElementAttribute(webservice.ext, elementname, elementtype, attributename, power)) //set the PV Installation production to 40kW

Console.WriteLine(string.Format("Output of {0} set to {1}kW", elementname, power));

else

{

CalcOk = false;

Console.WriteLine("Could not change output of " + elementname);

}

}

internal void RepeatLoadFLowandResults(Webservice webservice)

{

if (!CalcOk)

return;

RunLoadFlow(webservice);

GetResultsLoadFlow(webservice);

}

internal void RestoreOriginalValues(Webservice webservice)

{

if (!CalcOk)

return;

string elementname = "PV Installation";

string elementtype = "ACDisperseGenerator";

string attributename = "Pset";

string power = "4";

webservice.nepService.SetElementAttribute(webservice.ext, elementname, elementtype, attributename, power);

elementname = "Verbraucher A";

elementtype = "Load";

short portNumber = 0;

webservice.nepService.SwitchElementAtPort(webservice.ext, elementname, elementtype, portNumber, true);

Console.WriteLine("Restored original values of the example");

}

internal void RunSwitchingOptimisation(Webservice webservice)

{

if (!CalcOk)

return;

webservice.ext = webservice.nepService.GetProject("SwitchingOpimization", null, null, null); //get the project

if (webservice.ext == null)

Console.WriteLine("Cannot get SwitchingOptimization project");

//run switching optimisation.

AnalysisReturnInfo analysis = webservice.nepService.AnalyseVariant(webservice.ext, Guid.NewGuid().ToString(), "SwitchingOptimization", "Default", string.Empty, string.Empty, string.Empty);

if (analysis.ReturnInfo == 1)

Console.WriteLine("Switching Optimisation run successfully!");

else

{

Console.WriteLine("Could not run Switching Optimisation!");

CalcOk = false;

}

}

internal void GetResultsSwitchingOptimization(Webservice webservice)

{

if (!CalcOk)

return;

var results = webservice.nepService.GetAllElementResults(webservice.ext, "SwitchingOptimization"); //get the results of switching optimisation

if (results != null)

foreach (var res in results)

Console.WriteLine(string.Format("Element {0} was switched {1} on side {2}", res.Name, GetXMLAttribute(res.XMLdata, "SwitchFinal"), res.portNr));

}

/// <summary>

/// gets the value of an xml element

/// </summary>

/// <param name="result"></param>

/// <param name="xmlelement"></param>

/// <returns></returns>

private string GetXMLAttribute(string result, string xmlelement)

{

if (string.IsNullOrEmpty(result))

return null;

XmlDocument xmldoc = new XmlDocument();

xmldoc.LoadXml(result);

XmlNodeList nodeList = xmldoc.GetElementsByTagName(xmlelement);

if (nodeList == null || nodeList.Count != 1)

{

CalcOk = false;

return null;

}

else

return nodeList[0].InnerText;

}

}

public class Webservice

{

public NeplanServiceClient nepService;

public ExternalProject ext;

private string username = "";

private string password = "";

public string project = "Neplan Beispiel";

public Webservice()

{

nepService = new NeplanServiceClient(); //instantiates the neplaservice

nepService.ClientCredentials.UserName.UserName = username; //give the username

nepService.ClientCredentials.UserName.Password = getMd5Hash(password); //give the password

try

{

nepService.Open(); //open the service

Console.WriteLine("Opened service");

ext = nepService.GetProject(project, null, null, null); //get the project

if (ext != null)

Console.WriteLine("Got project");

else

Console.WriteLine("Cannot get project");

}

catch

{

Console.WriteLine("Cannot open service");

}

}

public void CloseWebservice()

{

try

{

nepService.Close(); //close the service

Console.WriteLine("Closed service");

}

catch

{

Console.WriteLine("Cannot close service");

}

}

private static string getMd5Hash(string input)

{

#region Not important for the training

// Create a new instance of the MD5CryptoServiceProvider object.

MD5 md5Hasher = MD5.Create();

// Convert the input string to a byte array and compute the hash.

byte[] data = md5Hasher.ComputeHash(Encoding.Default.GetBytes(input));

// Create a new Stringbuilder to collect the bytes

// and create a string.

StringBuilder sBuilder = new StringBuilder();

// Loop through each byte of the hashed data

// and format each one as a hexadecimal string.

for (int i = 0; i < data.Length; i++)

sBuilder.Append(data[i].ToString("x2"));

// Return the hexadecimal string.

return sBuilder.ToString();

#endregion

}

}

}